Лабораторная работа №8. Beans, JSP, JSF

Теоретическая часть

Технология Java Beans

Один из распространённых подходов программной инженерии «компонентное программирование» предполагает создание сложных систем на основе более простых программных компонентов. Такие компоненты проще проектировать, кодировать, отлаживать, распространять и модифицировать. Компонент может быть создан в рамках какого-либо проекта, а затем многократно использован в других проектах.

В языке Java реализацией идей компонентной архитектуры являются bean-компоненты. Он не имеет ограничений в плане функциональности, может быть визуальным (например, кнопка графического интерфейса) либо скрыт от пользователя в недрах системы (пример, декодер видеопотока в конкретном формате), может предназначаться для автономной работы на рабочей станции пользователя либо взаимодействовать с другими компонентами распределённой программной системы. Программная система, использующая bean-компонент, называется контейнером.

Основными достоинствами bean-компонентов являются следующие:

− компонент поддерживает основную парадигму Java «написано однажды, работает везде»;

− можно управлять свойствами, событиями и методами компонента, доступными другому приложению;

− программное обеспечение, обеспечивающее конфигурирование компонентов, не обязательно включать в среду времени выполнения;

− настройки параметров компонента можно хранить отдельно, восстанавливая их по необходимости;

− компонент может регистрироваться на получение событий о других объектов, и может генерировать события, отправляемые другим объектам.

Bean-компоненты обеспечивают эффективную реализацию других составных частей платформы Java, в том числе Swing, JSP и JSF.

Функциональные возможности bean-компонентов обеспечивается классами и интерфейсами пакета java.beans.

Рассмотрим основные моменты технологии Java Beans.

Самодиагностика компонента

В основе технологии Java Beans лежит свойство самодиагностики. Самодиагностика – это процесс анализа bean-компонента, при котором определяются его характеристики. С её помощью любое приложение, например среда разработки, может получать информацию о компоненте.

Существует два способа предоставления такой информации. В простейшем случае разработчик компонента просто следует определённым соглашениям об именовании свойств, событий и методов (методика проектных шаблонов). Анализируя имена, механизм самодиагностики находит все общедоступные методы bean-компонентов, игнорируя защищённые и приватные методы.

Свойство bean-компонента описывает некоторый аспект его состояния. Присваиваемые свойствам значения определяют поведение компонента. Все внутренние поля, реализующие хранение свойств, должны быть скрыты. Настройка свойства осуществляется посредством метода записи, а его получение посредством метода чтения.

Для методов, связанных со свойствами, в Java существуют специальные шаблоны именования.

Свойства могут быть простыми и индексированными. Простое свойство имеет только одно значение. Его можно идентифицировать с помощью следующего проектного шаблона, в котором N – это имя свойства, а T – его тип:

```
public T getN( );
public void setN(T arg);
```

Отметим, что свойство, доступное только для чтения или только для записи не будет иметь соответственно метода записи или метода чтения.

Индексированное свойство состоит из нескольких значений и идентифицируется посредством следующих двух проектных шаблонов:

```
public T getN(int index) /* 1 –Шаблон для доступа к одному */
public void setN(int index, T arg) /* конкретному значению свойства */
public T[] getN() /* 2 – Шаблон для доступа сразу */
public void setN(T values[]) /* ко всем значениям свойства */
```

Bean-компоненты используют рассмотренную на предыдущих лекциях модель делегирования событий. События могут идентифицироваться с помощью следующего шаблона, в котором T представляет тип события:

```
public void addTListener(TListener eventListener)
public void addTListener(TListener eventListener)
public void removeTListener(TListener eventListener)
```

Два первых метода служат для добавления слушателей указанного события. Первый используется в случае, если на получение уведомления о событии может быть подписано несколько слушателей. Второй – если событие допускает только одного слушателя. Последний метод шаблона удаляет слушателей.

Проектные шаблоны неявно определяют доступную пользователю компонента информацию. Второй способ самодиагностики предоставляет искомую информацию в явном виде, для чего создаётся дополнительный информационный класс, расширяющий специальный интерфейс BeanInfo.

Основными методами BeanInfo являются следующие:

```
PropertyDescriptor[] getPropertyDescriptors()
EventSetDescriptor[] getEventSetDescriptors()
MethodDescriptor[] getMethodDescriptors()
```

Реализуя эти методы, разработчик должен «создать» массивы объектов, содержащих информацию соответственно о свойствах, событиях и методах bean-компонента, доступных пользователю.

Отметим, для bean-компонента с именем MyBeanExample информационный класс должен называться MyBeanExampleBeanInfo.

Если нет необходимости реализовывать все методы интерфейса BeanInfo, можно унаследовать информационный класс от класса SimpleBeanInfo. Последний обеспечивает стандартную реализацию интерфейса BeanInfo, включая и три перечисленных метода. Например, если не переопределить в SimpleBeanInfo метод getPropertyDescriptors, то для определения свойств bean-компоненте будет использоваться самодиагностика проектного шаблона.

Контроль состояния компонента

Bean-компонент может иметь связанные и ограниченные свойства. Во время изменения связанного (bound) свойства, компонент генерирует событие типа ProprtyChangeEvent. Класс, осуществляющий обработку этого события должен реализовывать интерфейс PropertChangeListener.

Такое же событие генерируется компонентом при попытке изменить значение ограниченного свойства. Объекты, предварительно зарегистрировавшиеся на уведомление о данном событии, могут отклонить предложенное изменение, сгенерировав исключение PropertyVetoException. Классс лушатель должен реализовывать интерфейс VetoableListener.

Сохранение состояния компонента

Способность bean-компонента сохранять своё состояние, включая значения свойств и локальные переменные экземпляров, на энергонезависимом устройстве и извлекать его по мере необходимости называется постоянством компонента (persistence). Для обеспечения постоянства компонента используются возможности сериализации, которые предлагают библиотеки классов Java. Самый простой способ – реализовать компонентом или одним из его предков маркерный интерфейс java.io.Serializable. В этом случае сериализация будет выполняться автоматически. При этом можно отключить сохранение отдельных полей компонента, указав для них ключевое слово transient.

Существует возможность самостоятельно реализовать сериализацию компонента, например, реализовав интерфейс java.io.Extenlizable.

Важно – сохранение конфигурации контейнером возможна только для тех bean-компонентов, в которых реализована поддержка сериализации.

Пример bean-компонента

Компонент будет отображать цветной прямоугольник с прямыми либо скруглёнными углами в соответствии с установленными пользователем свойствами.

```
import java.awt.*;
import java.awt.event.*;
import java.io.Serializable;

public class MyFigures extends Canvas implements Serializable {
 transient private Color color; // Не сериализуется
 private boolean rectangular;
 public MyFigures() {
  addMouseListener(new MouseAdapter() {
   public void mousePressed(MouseEvent me) {change();}
  });
  rectangular = false;
  setSize(200, 100);
  change();
 }
 public boolean getRectangular(boolean flag) {return rectangular;}
 public void setRectangular(boolean flag) {
  this.rectangular = flag;
  repaint();
 }
 public void change() {
  color = randomColor();
```

```
 repaint();
 }
 private Color randomColor() {
  int r = (int)(255 * Math.random());
  int g = (int)(255 * Math.random());
  int b = (int)(255 * Math.random());
  return new Color(r, g, b);
 }
 public void paint(Graphics g) {
  Dimension d = getSize();
  int h = d.height;
  int w = d.width;
  g.setColor(color);
  if (rectangular) {
   g.fillRect(0, 0, w-1, h-1);
  }
  else {
   g.fillOval(0, 0, w-1, h-1);
  }
 }
 public static void main(String[] args){
  Frame f = new Frame();
  f.setLayout(new GridLayout());
  Panel panel = new Panel();
  panel.add(new MyFigures());
  f.add(panel);
  f.setVisible(true);
 }
}
```

Для реализации явной самодиагностики компонента воспользуемся классом SimpleBeanInfo.

```
import java.beans.*;

public class MyFiguresBeanInfo extends SimpleBeanInfo {
 public PropertyDescriptor[] getPropertyDescriptors() {
  try {
        PropertyDescriptor rectangular = new PropertyDescriptor("rectangular", MyFigures.class);
   PropertyDescriptor pd[] = {rectangular};
   return pd;
  }
  catch(Exception e) {System.out.println("Возникло исключение: " + e);}
  return null;
 }
 public static void main(String[] args){
  MyFiguresBeanInfo mf = new MyFiguresBeanInfo();
  mf.getPropertyDescriptors();
 }
}/*Возникло исключение: java.beans.IntrospectionException: Method not found: isRectangular*/
```

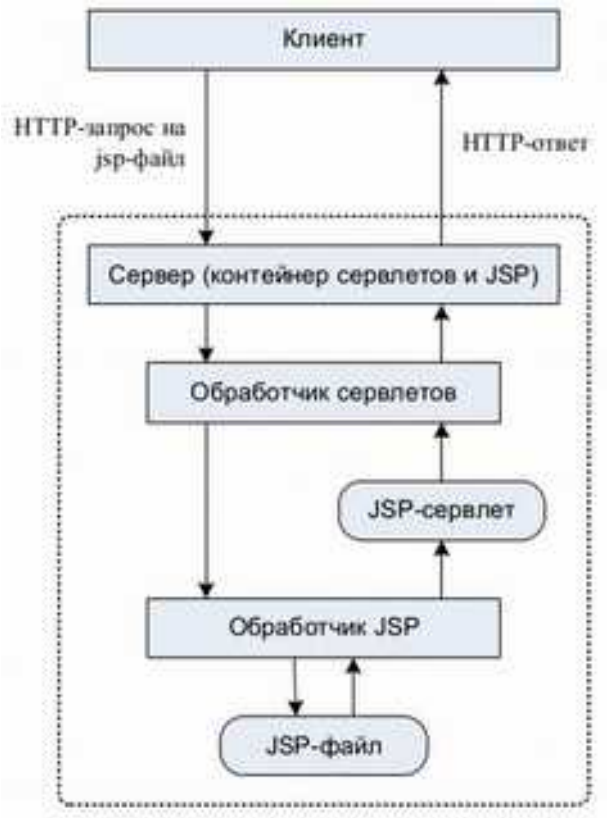
Обратите внимание, поскольку класс MyFiguresBeanInfo заменяет метод getPropertyDescriptors, то единственным свойством, которое может «увидеть» контейнер, является rectangular. Однако другие методы, например, getEventDescriptors здесь не переопределяются, поэтому будет использована самодиагностика проектного шаблона. Тем самым контейнеру будут доступны все события, включая те, что принадлежат его суперклассу Canvas.

Введение в технологию JavaServer Pages

Разработка технологии сервлетов позволила повысить качество разработки серверно-ориентированных веб-приложений. Однако эта технология имеет два существенных недостатка: во-первых, процесс разработки ненагляден, а, во-вторых, разработчик должен иметь высокую квалификацию и хорошее знание языка Java. Решением данной проблемы стала новая технология платформы Java, получившая название JavaServer Pages (JSP).

Для создания JSP-документа используется обычный язык разметки html, что снижает требования к квалификации web-разработчика. С другой стороны технология JSP основана на технологии сервлетов, и соответственно JSP-страница может содержать практически любой код на языке Java, что позволяет web-разработчику использовать всю мощь платформы Java, включая различные библиотеки.

JSP-документ размещается на web-сервере, как и обычная html-страница. Отличительным признаком будет расширение: jsp. Единственное требование, сервер должен поддерживать обе технологии – сервлеты и JSP. Эталонным, но не единственным сервером, реализующим эти требования является Apache Tomcat.



Когда клиент отправляет запрос на JSP-документ, сервер находит его, преобразует в сервлет, а затем передаёт управление полученному сервлету.

JSP-страница компилируется в сервлет при первом обращении. При следующих запросах сервер будет обращаться уже непосредственно к сервлету.

В отличие от технологии сервлетов, если JSP-страница была изменена, немедленная перекомпиляция не требуется. Она выполнится автоматически при следующем обращении клиента к странице. Перезапуск сервера также не требуется.

Архитектура web-приложения

Множество связанных JSP-страниц образуют web-приложение. Оно имеет ту же структуру служебных папок и файлов, что и приложение, основанное на сервлетах. Технология JSP предполагает, хотя и не навязывает разработку web-приложений в рамках компонентно-ориентированной архитектуры. Данная архитектура называется странице-центричной (Model 1) и наиболее распространена.

В основе альтернативного подхода (Model 2) лежит шаблон проектирования Модель-Представление-Поведение (Model-View-Controller), что предполагает разделение функции представления данных приложения пользователю (View, Presentation layer) и функции управления потоком исполнения приложения (Controller, Control layer) на отдельные компоненты web-приложения. Презентационный слой приложения реализуется с помощью JSP-страниц, а управляющий слой – в форме сервлетов (сервлет-центричность). Bean-компоненты отвечают за данные приложения (Model).

Основы синтаксиса JSP

В основе JSP лежит html-код, который может дополняться специальными JSP-тэгами:

<%= Java-выражение %> Внутри тэга записываются вычисляемые выражения на языке Java.

<% Java-код %> Данный тэг называется скриплетом. Внутри него может быть записан произвольный Java-код (декларация переменных, вычисление выражение, вызов функций и т.п.).

<%! Java-декларация %> Внутри тэга объявляются Java-переменные и Java-методы.

<% – – Текст комментария – – %> Данный тэг содержит комментарии. Естественно, при компиляции JSP-страницы в сервлет комментарии пропускаются и клиенту не передаются.

Предопределённые jsp-переменные

JSP-документ может содержать ссылки на следующие предопределённые системные переменные:

request – Представляет объект типа HttpServletRequest, обеспечивая доступ к различной информации, связанной с текущим запросом. Например, можно узнать значение параметра запроса с именем MyParam: request.getParameter(“MyParam”).

response – Представляет объект типа HttpServletResponse, обеспечивая возможность управлять ответом сервера а текущий запрос. Например, можно выполнить явную переадресацию клиента на страницу с адресом MyNewURL: response.sendRedirect(“MyNewURL”).

out – Представляет объект типа JspWriter. Этот объект является буферизованной версией класса PrintWriter и обеспечивает возможность создавать html-код в теле скриплетов.

application – Представляет объект типа ServletContext, обеспечивая доступ к различным данным, актуальным в контексте всего web-приложения.

session – Представляет объект типа HttpSession, обеспечивая доступ к различным данным, актуальным в контексте текущей сессии клиента.

config – Представляет объект типа ServletConfig, обеспечивая доступ к параметрам инициализации, использовавшимся при запуске JSP-сервлета.

pageContext – Представляет объект типа PageContext. Данный объект обеспечивает доступ к атрибутам текущей JSP-страницы, позволяет клиенту сохранять данные в различных контекстах, а также управляет обработкой ошибок.

page – Представляет ссылку на текущий экземпляр JSP-сервлета. В технологии сервлетов аналогичную роль выполняет объект this, используемый в методах класса сервлета.

Директивы

Группа тэгов, называемых директивами, помогает управлять структурой итогового JSP-сервлета. Директивы могут располагаться в любом месте, но, желательно, располагать их в начале страницы.

Директива page используется для установки атрибутов, влияющих на всю JSP-страницу целиком. Она имеет следующий синтаксис (соответственно в обычном и xml-формате):

```
<%@ page attribute=value %>
<jsp:directive.page attribute=value />
```

Как правило, по умолчанию JSP-сервлет импортирует пакеты java.lang.*, javax.servlet.*, javax.servlet.jsp.* и javax.servlet.http.*. Некоторые веб-серверы расширяют этот список. Разработчик может явно указать на необходимость импорта в JSP-сервлет определённого пакета с помощью директивы page и её атрибута import, например:

```
<%@ page import=”java.util.*” %>
```

У page существуют и другие атрибуты. Отметим, что import – единственный из них, который можно использовать в одном JSP-документе более одного раза.

Процесс разработки JSP-страниц можно упростить в тех случаях, когда различные страницы содержат одинаковые фрагменты JSP-кода. Существует два способа включить файл в JSP-страницу.

Первый способ основан на директиве include:

```
<%@ include file=”URL-ссылка на включаемый файл” %>
<jsp:directive.include file=”URL-ссылка на включаемый файл” />
```

Содержимое файла, указанного в директиве include будет статически включено в основной файл. Если вложенный файл впоследствии будет изменён, это никак не скажется на работе web-приложения. Потребуется принудительно вызвать перекомпиляцию основного JSP-файла. Для большого приложения данная задача может оказаться чрезмерно трудоёмкой.

Второй способ использует следующий тэг:

```
<jsp:include page=”URL-ссылка на включаемый файл” />
```

Содержимое указанного в тэге файла включается в основной файл динамически во время выполнения запроса клиента. Любые изменения во включаемом файле будут учитываться сервером, при этом перекомпиляция основного файла в JSP-сервлет не

требуется. Как следствие включаемый файл не может содержать JSP-код, поскольку он требует компиляции.

Директива jsp:plugin используется для включения в JSP-страницу апплетов. Директива выступает в качестве альтернативы таким html-тэгам как <applet>, <object> и <embed>.

JSP и компоненты Java beans

Технология JSP предполагает, хотя и не навязывает разработку web-приложений в рамках компонентно-ориентированной архитектуры. Данная архитектура использует bean-компоненты для удаления java-кода из JSP-файлов. Поскольку Java Bean является полноценным компонентом, к нему могут обращаться различные JSP-страницы. Более того, разработчику страницы достаточно знать лишь о существовании bean-компонента, знание внутренней реализации компонента ему абсолютно не нужно. Таким образом можно эффективно разделить разработку web-приложения между java-разработчиками и web-дизайнерами.

Основная идея связки JSP — JavaBean состоит в минимизации связей между файлом страницы и файлом компонента, поэтому основными инструментами связки стали три следующих тэга:

<jsp:useBean> – Используется для создания внутри страницы экземпляра bean-объекта заданного класса либо получения ссылки, если такой объект уже существует;

<jsp:getProperty> – Обеспечивает чтение свойства компонента.

<jsp:setProperty> – Обеспечивает модификацию свойства компонента.

Основные атрибуты тэга <jsp:useBean>: атрибут id задаёт имя объекта, а атрибут class указывает имя соответствующего класса. Время, в течение которого будет существовать bean-компонент определяется атрибутом scope.

Два оставшихся тэга используют атрибут name – для задания имени экземпляра bean-объекта, property – для задания имени свойства и value – для указания нового значения свойства. Поскольку bean-компонент является Java-классом, его свойства могут иметь любой примитивный или классовый тип.

Однако для упрощения связки JSP-Java Bean все они автоматически будут заменены в JSP-файле типом String. Преобразование типов выполняется JSP-контейнером при использовании методов чтения и записи свойств bean-компонента.

После того, как будет создан либо просто получен экземпляр bean-компонента, можно использовать его в скриплетах и в Java-выражениях.

Язык выражений

Позволяет уменьшить явное применение Java-кода. Этот механизм предоставил JSP-разработчикам новый язык записи выражений, а потому получил название «язык выражений» (expression language, EL). Первоначально язык выражений можно было использовать только в атрибутах тэгов. Текущая версия стандарта JSP позволяет использовать его по всей странице.

Отметим, применение языка выражений не вступает в конфликт с JSP-тэгами выражений.

Общий синтаксис выражений:

${expression},

где expression – любая конструкция, соответствующая правилам языка.

Если выражение используется в качестве атрибута, его необходимо записывать как строку – в окаймляющих двойных кавычках. Пример.

```
...
<h1>Пример 3.3.5 - Bean-компонент</h1>
<jsp:useBean id="myBean" scope="page"
class="kai.tci.java_lesson.MyBean" />
<jsp:setProperty name="myBean" property="a" value="1" />
<p>Исходное значение переменной a = ${myBean.a} /></p>
<%-- Вместо <% myBean.makeFirstChange(); %> --%>
<jsp:setProperty name="myBean" property="a" value="${myBean.a – 1}" />
<p>Следующее значение переменной a = ${myBean.a} /></p>
<%-- Вместо <% myBean. makeSecondChange (); %> --%>
<jsp:setProperty name="myBean" property="a" value="${myBean.a + 2}"/>
<p>Итоговое значение переменной a = ${myBean.a} /></p>
...
```

Здесь выполнено два типа изменений. Во-первых, обращение к достаточно громоздкому тэгу jsp:getProperty заменено компактным EL-выражением ${myBean.a}. Во-вторых, продемонстрирован альтернативный скриплету способ изменения значений свойства bean-компонента с использованием EL-выражений и тэга jsp:setProperty.

Язык выражений включает в себя следующие синтаксические элементы:

1) константы (логические: true и false, целочисленные, вещественные, строковые и null);

2) операторы «[]» и «.» для доступа соответственно к индексированным и именованным свойствам.

3) операции (арифметические, логические и операции отношения);

4) оператор empty.

Оператор empty проверяет операнд на «пустоту». Например, для строковой переменной s выражение ${empty s} эквивалентно Java-условию:

```
if ((s != null) && (s.length() > 0))
```

Для целочисленной переменной выражение ${empty m} эквивалентно условию:

```
if ((m != null) && (m != 0))
```

Язык выражений определяет множество неявно создаваемых объектов. Например, выражение ${pageContext} возвращает системный объект типа PageContext. Подробное изучение перечня таких объектов выходит за рамки данного курса. С ними можно ознакомиться в документации на стандарт JSP.

Связь JSF и JSP

Каркас JSF реализует компонентно-ориентированную модель (Model2): данные модели управляются bean-компонентами. Функции управления реализуются автоматически генерируемым сервлетом.

Интерфейс JSF-приложения состоит из JSP-страниц, которые содержат компоненты, обеспечивающие функциональность интерфейса. При этом библиотеки тегов JSP используются на JSP-страницах для отрисовки компонентов интерфейса, регистрации обработчиков событий, связывания компонентов с валидаторами и конвертерами данных и т.п.

Тем не менее, JSF не привязана жёстко к JSP: она использует JSP через специальную библиотеку тегов. Данные теги реализуют лишь «отрисовку» компонентов, обращаясь к ним по имени. Жизненный же цикл компонентов JSF не ограничивается JSP-страницей.

Пример использования beans, JSP, JSF

jsp_1.jsp

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>

<%-- Объявление библиотек тэгов JSF --%>
<%@taglib prefix="f" uri="http://java.sun.com/jsf/core"%>
<%@taglib prefix="h" uri="http://java.sun.com/jsf/html"%>

<!DOCTYPE html>
<html xmlns:h="http://xmlns.jcp.org/jsf/html" xmlns:f="http://xmlns.jcp.org/jsf/core">
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>Словарь-переводчик</title>
   </head>
   <body>
      <jsp:useBean id="mybean" scope="session" class="jspappl.NameHandler" />
      <h3>Введите ваше имя</h3>
      <form name="Input form" action="jsp_2.jsp">
         <input type="text" name="name" />
         <input type="submit" value="OK" name="button1" />
         <%mybean.addCounter(1);%>
      </form>
      <a href="jsp_2.jsp?name=123">Переход</a>
   </body>
</html>
```

jsp_2.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>JSP Page</title>
   </head>
   <body>
      <%! int counter; %>
      <jsp:useBean id="mybean" scope="session" class="jspappl.NameHandler" />
      <%mybean.addCounter(Integer.parseInt(request.getParameter("name")));%>
      <jsp:setProperty name="mybean" property="name" />
      <h3>Приветствую тебя, <jsp:getProperty name="mybean" property="name" />, так.</h3>
      <h3>Приветствую тебя, ${mybean.name}, и вот так. </h3>
      <form name="Back form" action="jsp_1.jsp">
         <input type="submit" value="Back" name="button2" />
      </form>
      <h3>Счётчик <jsp:getProperty name="mybean" property="counter" /></h3>
      <h3>Счётчик <%=++counter %></h3>
```

```
    </body>

</html>
```

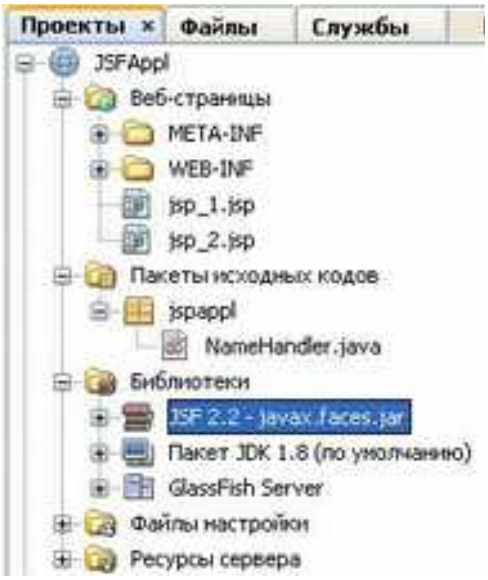
NameHandler.java

```
package jspappl;

public class NameHandler {
    private String name;
    private int counter;
    public NameHandler(){
        name = null;
        counter = 0;
    }

    /**
     * @return the name
     */
    public String getName() {
        return name;
    }

    /**
     * @param name the name to set
     */
    public void setName(String name) {
        this.name = name;
    }

    /**
     * @return the counter
     */
    public int getCounter() {
        return counter;
    }

    /**
     * @param counter the counter to set
     */
    public void setCounter(int counter) {
        this.counter = counter;
    }

    public void addCounter(int i) {
        this.counter = this.counter + i;
    }

}
```

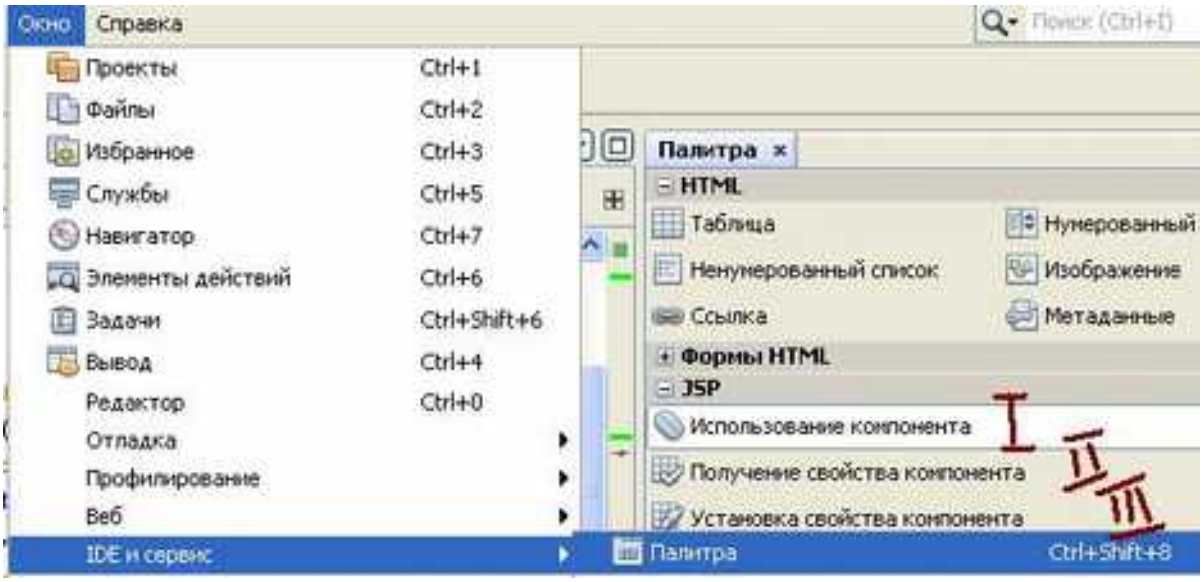
Полезные свойства проекта

Свойства проекта\Контекстный путь = /JSFAppl
Свойства проекта\Относительный URL-адрес = jsp_1.jsp

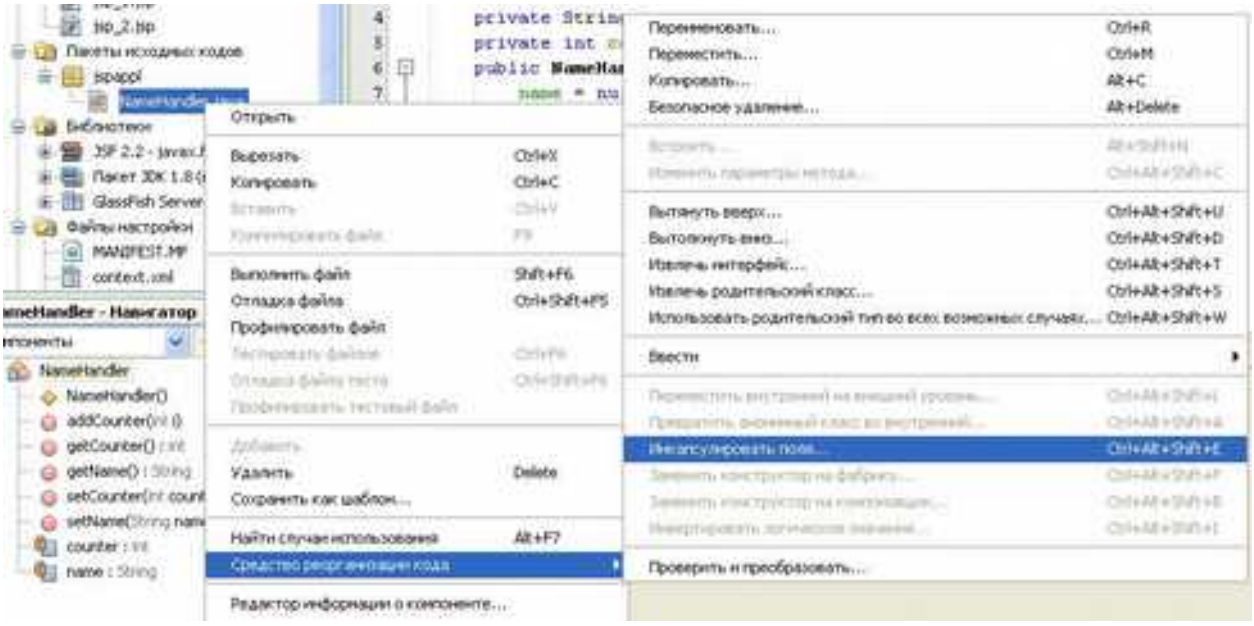
Далее приводится примерный внешний вид вашего проекта в браузере IDE NetBeans.

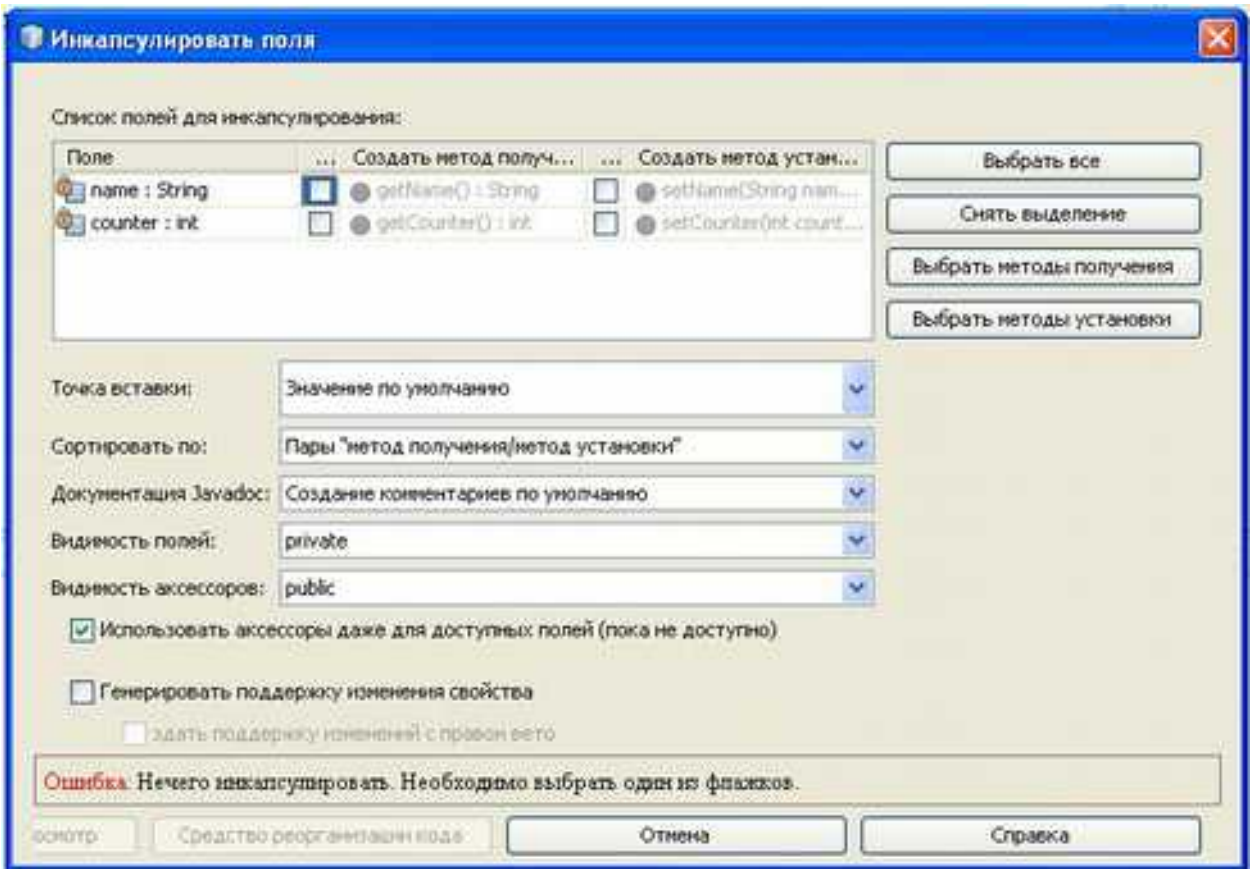


Далее приведены полезные свойства палитры (инструмента IDE NetBeans), которые позволяют внедрить bean — компоненту в JSP — страницу, получить свойство bean — компоненты, расположенной на JSP — странице, и установить свойство bean — компоненты, находящейся на странице.



Для автоматизации создания bean — компонент вы можете использовать средства реорганизации кода, например «инкапсуляция полей». Смотрите следующие два рисунка.


Открыть
Вырезать Ctrl+X
Копировать Ctrl+C
Вставить Ctrl+V
Выполнить файл Shift+F6
Отладка файла Ctrl+Shift+F5
Профилировать файл
Удалить Delete
Сохранить как шаблон...
Найти случаи использования Alt+F7
Средство реорганизации кода
Редактор информации о компоненте...
Переименовать... Ctrl+R
Переместить... Ctrl+M
Копировать... Alt+C
Безопасное удаление... Alt+Delete
Вытянуть вверх... Ctrl+Alt+Shift+U
Вытолкнуть вниз... Ctrl+Alt+Shift+D
Извлечь интерфейс... Ctrl+Alt+Shift+T
Извлечь родительский класс... Ctrl+Alt+Shift+S
Использовать родительский тип во всех возможных случаях... Ctrl+Alt+Shift+W
Ввести
Инкапсулировать поля... Ctrl+Alt+Shift+E
Проверить и преобразовать...


Инкапсулировать поля
Список полей для инкапсулирования:
Поле
Создать метод получ...
Создать метод устан...
name : String
counter : int
Выбрать все
Снять выделение
Выбрать методы получения
Выбрать методы установки
Точка вставки:
Значение по умолчанию
Сортировать по:
Пары "метод получения/метод установки"
Документация Javadoc:
Создание комментариев по умолчанию
Видимость полей:
private
Видимость аксессоров:
public
Использовать аксессоры даже для доступных полей (пока не доступно)
Генерировать поддержку изменения свойства
Ошибка: Нечего инкапсулировать. Необходимо выбрать один из флажков.
Отмена
Справка

Практическая часть

1. В зависимости от варианта выполняется разработка web-приложения. Пример приложения похожего на то, которое нужно разработать, см. выше. Показать работу приведённого выше примера.
2. В целом, нужно создать web-приложение в NetBeans IDE, которое должно состоять из трёх JSP-страниц – стартовой, главной и финишной, а также при необходимости использовать для обмена данными между страницами Bean-компоненту.
3. На «Главной странице» инициировать вычисление функции из задания первой лабораторной работы.
4. Параметры необходимые для работы функции реализуемой на «Главной странице» задавать на ней в текстовом поле, а результат её работы показывать на «Финишной странице».
5. Программный код вычисляемой функции разместить: 0 - на «Главной странице», 1 – в классе Bean-компоненты.
6. Заголовки страниц должны иметь следующий вид: 0 – «Стартовая страница», «Главная страница» и «Финишная страница»; 1 - «Раз», «Два» и «Три».
7. Формат «Стартовой страницы»: 0 – содержит текст задания на лабораторную работу, ФИО студента и ссылку для перехода на «Главную страницу», 1 – содержит текст задания на лабораторную работу, группа студента и кнопку для перехода на «Главную страницу».
8. Организовать вывод результатов работы функции на «Финишной странице»: 0 - строки следуют одна за другой сверху вниз (таблица без видимых границ, состоящая из одного столбца и множества строк); 1 – результаты должны быть каким-то образом размещены в видимой таблице, в таблице допускается произвольное число столбцов и строк.
9. При повторном переходе на «Главную страницу», например при нажатии кнопки «Возврат» на «Финишной странице», на «Главной странице» должно отображаться: 0 – изменение триггера, размещённого на «Главной странице»; 0 – изменение триггера, размещённого в Bean-компоненте; 1 – изменение счётчика, размещённого на «Главной странице»; 2 – изменение счётчика, размещённого в Bean-компоненте; 3 – общее число переходов (обновлений) страниц Web-приложения.
10. Среди средств автоматизации разработки web-приложения можно выделить инструмент окно «Палитра» для JSP – страниц и средства реорганизации кода для класса bean-компоненты, см. рисунки выше.

Варианты заданий

Таблица №1

№	(П.5)	(П.6)	(П.7)	(П.8)	(П.9)
1.	1	0	1	1	11

Число в указанном варианте следует толковать как двоичное число, каждый разряд которого соответствует столбцу из таблицы 1, кроме столбца «П.9» — для него задействованы два последних разряда указанного числа.

1	**47(101111)**	17	49	33	56	49	21
2	1	18	28	34	53	50	8
3	4	19	29	35	46	51	3
4	12	20	38	36	62	52	8
5	12	21	34	37	18	53	64
6	37	22	37	38	29	54	28
7	54	23	32	39	26	55	10
8	53	24	34	40	4	56	15
9	46	25	13	41	15	57	5
10	38	26	47	42	8	58	35
11	35	27	37	43	56	59	16
12	17	28	21	44	14	60	49
13	12	29	40	45	33	61	21
14	37	30	57	46	50	62	9
15	52	31	61	47	37	63	60
16	24	32	60	48	32	64	51